ECE 109-001 Program 2: `iworm.asm`

Fall 2019

Flowchart Due on Moodle October 15, 2019

Due Monday October 21 @ 11:45pm

This programming assignment must be completed individually. Do not share your code with or receive code from any other student. The only people who may see your code are the instructor and the ECE 109 TAs.

Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of -100 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

For this assignment, you will write a program that draws lines on PennSim's graphic display, similar to an Etch A Sketch®. The program user will be able to "drag" a small box in an up-down or left-right direction to draw a line, and will be able to change the color of the line that is being drawn.

The learning objectives for this assignment are:

- Use load and store instructions to manipulate the content of memory.
- Use I/O routines to allow a user to interact with the program.

Program Specification

The program must start at address x3000.

The program will manipulate the location of a 2x2 box on the screen, which we will call the Pen. The Pen has a color and a location. The screen pixel at the Pen's current location will take on the Pen's color. When the Pen moves, the pixels at the previous location retain their color. In other words, moving the Pen will draw a line of a particular color.

The PennSim graphics display (the "screen") is 128 by 124 pixels. We use an (x, y) coordinate system to describe a location on the screen. Location (0, 0) is the top left corner. The x coordinate increases as we move to the right, and the y coordinate increases as we move down. In other words, (1, 0) is one pixel to the right of (0, 0), and location (0, 1) is one pixel below (0, 0). Location (127, 123) is the bottom right corner of the screen.

When the program begins, the Pen location must be set to (64, 62) and the color must be White.

The user interacts with the program using one-character commands. The commands are typed on the keyboard, but are not echoed to the console display. Nothing will be printed to the console during the

execution of this program. The program will wait for a keystroke, perform the corresponding command, and repeat. If the keystroke does not correspond to a legal command, it will have no effect.

There are four commands for changing the location of the Pen. We use the WASD scheme of navigation, used by various computer games:

Command Character	Action
w	*Move up two pixels.* Location changes from (x, y) to (x, y-2). If the Pen is at the top border of the screen, the command has no effect.
a	*Move left two pixels.* Location changes from (x, y) to (x-2, y). If the Pen is at the left border of the screen, the command has no effect.
s	*Move down two pixels.* Location changes from (x, y) to (x, y+2). If the Pen is at the bottom border of the screen, the command has no effect.
d	*Move right two pixels.* Location changes from (x, y) to (x+2, y). If the Pen is at the right border of the screen, the command has no effect.

There are five commands for changing the Pen color:

Command Character	Action
r	Change color to *Red*.
g	Change color to *Green*.
b	Change color to *Blue*.
y	Change color to *Yellow*.
space	Change color to *White*.

Note: When you change the color of the Pen, the color of the current location must change, before the next command is processed.

There are two additional commands:

Command Character	Action
return	*Clear the screen.* Paint all pixels black, except the Pen location, which retains its color.
q	*Quit.* The simulated machine must stop running.

Details

The PennSim graphics display is bit-mapped, meaning that each pixel has a corresponding memory location. The content of that memory location controls the color of the pixel.

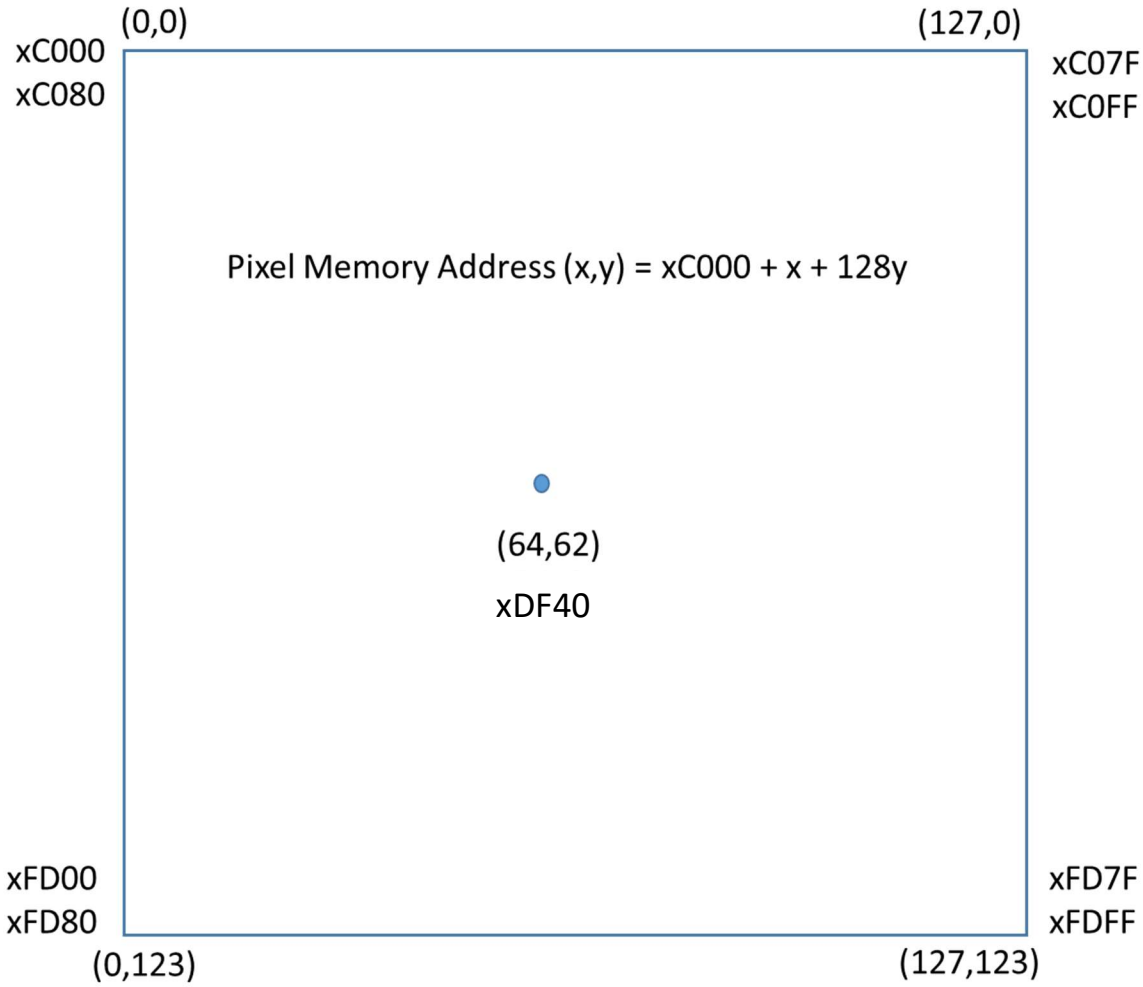
Pixel Addresses

Addresses xC000 through xFDFF are assigned to the graphics display. The low address corresponds to the top left corner (0, 0). Moving one pixel to the right adds one to the address, and it "wraps around" to the next row when it gets to the right edge. Since the display is 128 pixels wide, this means that moving down one pixel is equivalent to adding 128 to the address.

The address of point (x, y) can be calculated as: xC000 + x + 128y.

For this assignment, you will not need to calculate arbitrary pixel addresses, except to figure out where the initial location (64, 62) is. You will be moving left (-2), right (+2), up (-256) or down (+256) from the current address.

You will, however, need to recognize when the Pen is at an edge of the display, so that you don't go beyond the edge.



Pixel Color

As mentioned above, the value of the pixel address determines the color of the pixel. The 16 bits contain 5 bits for each RGB component of color: bits [14:10] for red, [9:5] for green, and [4:0] for blue. Bit 15 is ignored. The higher value of a component, the more of that color is present. The table below gives the color values (in hex) needed for this program.

Color	Value
Red	x7C00
Green	x03E0
Blue	x001F
Yellow	x7FED
White	x7FFF
Black	x0000

Miscellaneous

For more explanation about the PennSim display, see the PennSim Reference Manual.

The ASCII code for the Return key is x0A (#10). This is listed as linefeed (LF) in the ASCII table.

Hints and Suggestions

- As always, **design before you code**! Draw a flowchart to organize and document your thinking before you start writing the program.
- **Work incrementally!** For example, implement one command at a time. Make sure the program works before moving on to the next command. This way, you always have working code.
- It's not a bad idea to submit each working version of your program to Moodle. Then, if your machine crashes (it happens!), you haven't lost everything. Each time you submit, it overwrites the previous submission, so you can submit as many times as you like. But don't expect that we can recover some previous version of your code if you accidentally clobber it. (You should have some sort of backup system for your schoolwork, right?)
- *Test your program with a wide variety of inputs.* Make sure that you have tested the "corner cases," such as reaching the border of the display.
- Use the PennSim simulator and assembler. There are other simulators and assemblers out there, but there are some differences. Your program will be graded using PennSim, and no other tools will be used or considered. You must use the provided **p2os.obj** and **p2os.sym** files for the operating system.

Administrative Info

Any corrections or clarifications to this program spec will be posted on the **Discussion Forum**. It is important that you read these postings, so that your program will match the updated specification. (I recommend strongly that you subscribe to the forum, so that you will not miss any updates or corrections.)

What to turn in:

- E-mail your one-page flow chart to Moodle by October 15, 11:45 pm.
- Assembly Language Source file – it must be named **iworm.asm**. Submit via **Moodle** to the Program 2 assignment.

  The program will be graded by one of the TAs, and your grade will be posted in the Moodle gradebook. You will also get back some information about what you got wrong, and how points were deducted (if any). Use the "Retrieve Assignment" feature of Moodle to get that information.
- DO NOT submit .obj or .sym files. Do not submit a .txt file, a .doc file, or anything like that. It must be a simple text file with the proper .asm extension. If we are unable to open or interpret your file, you will get a zero for the assignment (even if it has the right name!).

Grading criteria:

5 points: Submission of Flow Chart by October 15.

5 points: Correct type of file, submitted with the **proper name**. (No partial credit!! These are essentially FREE POINTS! Don't screw it up.)

20 points: **Program is complete** and **assembles with no warnings and no errors** using the PennSim assembler. To be "complete," there must be code that makes a reasonable attempt to meet the program specs. Programs that do not assemble will not be graded any further. (For warnings, points will be deducted, but the program will be tested for correctness.)

10 points: **Proper coding style, comments, and header**. Use indentation to easily distinguish labels from opcodes. Leave whitespace between sections of code. Include *useful* comments and *meaningful* labels to make your code more readable. Your file **must** include a header (comments) that includes your name and a description of the program. Don't cut-and-paste the description from the program spec – that's plagiarism. Describe the program in your own words. This category is somewhat subjective, but the goal is that your program should be easy to read and to understand.

60 points: The program **handles all commands correctly**:

(30 points) WASD movement (7.5 pts each).
(20 points) Change colors (5 pts each).
(5 points) Clear display.
(5 points) Quit.